ALFAMA CURSOS

alfamacursos.com.br

Alfama Cursos

Antônio Garcez
Fábio Garcez
Diretores Geral

Antônio Álvaro de Carvalho
Diretor Acadêmico

MATERIAL DIDÁTICO

Produção Técnica e Acadêmica

Marcela Menezes Flores
Coordenadora Geral

Patrícia Queiroz de Meneses
Coordenadora Pedagógica

José Alves Correia Neto
Renata Jacomo Viana
Autoria

Gabriella Caroline Teles Silva
Sabina Regina Conceição Santos
Revisão Textual

Rafael Rezende de Farias
Editoração

Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/98.
É proibida a reprodução total ou parcial, por quaisquer meios, sem autorização prévia, por escrito, da ALFAMA CURSOS.

alfamacursos.com.br

Introdução a Algoritmos e Estrutura de Dados

Apresentação do Curso

Olá caro aluno!

Seja bem-vindo ao curso de Introdução a Algoritimos e Estrutura de Dados, sabendo que é uma matéria de extrema importância para o seu conhecimento. Esta apostila vem complementar as videoaulas.

Lembre-se que apenas pela prática será possível ser um bom programador.

Apresentação do Professor

FÁBIO GOMES ROCHA é formado em Sistemas da Informação e pós-graduado em Engenharia de Sistemas. Atualmente é membro da ABNT (Associação Brasileira de Normas Técnicas), fazendo parte do Comitê 21 - Engenharia de Software e do Comitê 27 - Segurança da Informação. Leciona Técnicas de Desenvolvimento de Sistemas e Linguagem de Programação e na Educação a Distância já trabalhou como desenvolvedor, professor autor e tutor em cursos universitários e técnicos, além de possuir publicações nesta área desde 2006. Além disso, atua na área de Tecnologia da Informação e Comunicação com Desenvolvimento de Sistemas há 17 anos.

Em 2004 publicou um livro sobre Tecnologia para Gestão, intitulado Secretariado: do Escriba ao Web Writer.

Componente Curricular

EMENTA

Introdução à Lógica; Fases de Desenvolvimento de um Programa; Algoritmo; Variáveis; Estrutura de Decisão; Algoritmo de Estrutura de Decisão; Estrutura de Repetição; Algoritmos com Estrutura de Repetição; Estrutura de Dados: Vetor; Estrutura de Dados: Matriz; Função; Função com Passagem de Parâmentro; Registro e Estrutura; Arquivos; Ordenação; Pesquisa; Listas; Pilhas; Fila; Árvores; Outras Estruturas; Ponteiro; Alocação Dinâmica e Orientação a Objetos.

PÚBLICO-ALVO:

Alunos oriundos do Ensino Médio e universitários que queiram complementar sua formação profissional.

OBJETIVOS GERAIS:

- Conhecer e desenvolver sistemas internos de acordo com as necessidades apontadas.

HABILIDADES E ATITUDES ESPERADAS:

- Aplicar conhecimentos básicos de construção e aplicação de algoritmos na programação de sistemas.
- Descrever estruturas básicas da linguagem.
- Aplicar formas de utilização dos arquivos.

Índice

1 - Introdução à Lógica 9
1.1 - Algoritmos 9
1.2 - Exercícios Propostos 11
2 - Fases do Desenvolvimento de um Programa 12
2.1 - Fases Clássicas 12
2.2 - Modelo Espiral 13
2.3 - Modelo Prototipagem 13
2.4 - Modelo Ágil 14
2.5 - Exercícios Propostos 16
3 - Algoritmo 17
3.1 - Algoritmo e Fluxograma 17
3.2 - Exercícios Propostos 21
4 - Variáveis 22
4.1 - Algoritmo e Variáveis 22
4.2 - Exercício Proposto 25
5 - Estrutura de Decisão 26
5.1 - Condicional 26
5.2 - Exercício Proposto 28
6 - Algoritmo de Estrutura de Decisão 29
6.1 - Condicional 29
6.2 - Exercício Proposto 32
7 - Estrutura de Repetição 33
7.1 - Exercício Proposto 34
8 - Algoritmo de Estrutura de Repetição 35
8.1 - Estrutura de Repetição 35
8.2 - Exercício Proposto 38
9 - Estrutura de Dados: Vetor 39
9.1 - Variáveis Compostas: Vetor 39
9.2 - Exercício Proposto 41
10 - Estrutura de Dados: Matriz 42
10.1 - Variáveis Compostas: Matriz 42
10.2 - Exercício Proposto 44
11 - Função 45
11.1 - Exercício Proposto 47
12 - Função com Passagem de Parâmetro 48
12.1 - Exercício Proposto 50
13 - Registro e Estrutura 51
13.1 - Exercício Proposto 52
14 - Arquivos 53
15 - Ordenação 55
15.1 - Métodos Simples 55
15.1.1 - Selection Sort 55
15.1.2 - Insertion Sort 55
15.1.3 - Bubble Sort 56
16 - Pesquisa 57
17 - Listas 59
17.1 - Lista de Compras 59
17.1.2 - Lista Ligada 59
18 - Pilhas 60
18.1 - Inclusão 60
18.2 - Remoção 60
18.3 - Uso 60
19 - Filas 61
19.1 - Inserção e Remoção 61
20 - Árvores 62

21 - Outras Estruturas 63
21.1 - Lista Duplamente Encadeada 63
21.2 - Lista Circular 63
21.3 - Grafos 63
22 - Ponteiro 64
23 - Alocação Dinâmica 65
24 - Orientação a Objetos 66
24.1 - O que é um Objeto? 66
Respostas dos Exercícios Propostos 67
Referências Bibliográficas 73

1 - INTRODUÇÃO À LÓGICA

O que é Lógica?

Segundo o Dicionário Aurélio é a "coerência de raciocínio, de ideias, sequência coerente, regular e necessária de acontecimentos, de coisas".
Exemplo:

- O número 5 é menor que o número 10.
- O número 15 é maior que o número 10.
- Logo, 5 é menor que 15. Lógico, não?

A lógica é tão antiga quanto à própria filosofia. Na verdade, a história da lógica se perde no tempo, sabe-se que ela foi utilizada em conjunto com a filosofia. Como exemplo temos a lógica dedutiva e a indutiva. Um exemplo aplicado de lógica é:

- João é brasileiro.
- Todo brasileiro é latino-americano.
- Logo, João é latino-americano.

Nota-se a sequência de raciocínio e de ideia sempre coerente. São inúmeros os exemplos, dos mais fáceis aos mais complexos, mas o importante é obedecer a uma sequência e coerência.

E o que é Lógica de Programação?

Lógica de Programação é a técnica de desenvolver algoritmos utilizando sequências lógicas para atingir um determinado objetivo.

A Lógica de Programação é a aplicação de coerência de raciocínio e sequência para o desenvolvimento de algoritmos.
Logo, programar é construir algoritmos utilizando a lógica. Um programa é um algoritmo codificado em uma linguagem de programação, isto é, um conjunto de instruções/funções que representam tarefas que serão interpretadas e executadas por um computador.

1.1 - ALGORITMOS

Um algoritmo na verdade é uma solução para um problema, criado de forma sequencial e coerente. O algoritmo é o conceito central da programação e da ciência da computação, ou seja, programar é criar algotirmos.

Etapas para a construção de um algoritmo:

- Entender o enunciado.
- Identificar operações necessárias.
- Organizar em sequência.
- Refletir sobre a solução.

Os algoritmos podem ser representados das seguintes formas:

- Linguagem natural **-->** depende de quem escreve e de quem lê.
- Linguagem gráfica **-->** símbolos padronizados.
- Pseudolinguagem **-->** próxima de uma linguagem de programação.
- Linguagem de programação **-->** é possível desenvolver o algoritmo diretamente na linguagem de programação desejada.

Como exemplo de algoritmos podemos citar:

- Os algoritmos utilizados nas operações básicas da Matemática (soma, subtração, divisão e multiplicação). Estes são bem conhecidos.

Um algoritmo é uma sequência finita de passos que levam à execução de uma tarefa, normalmente sendo a solução de um problema. Um exemplo básico de algoritmo é o procedimento de uma receita, na qual temos os ingredientes e os passos para fazer o bolo. Este passo é o algoritmo.

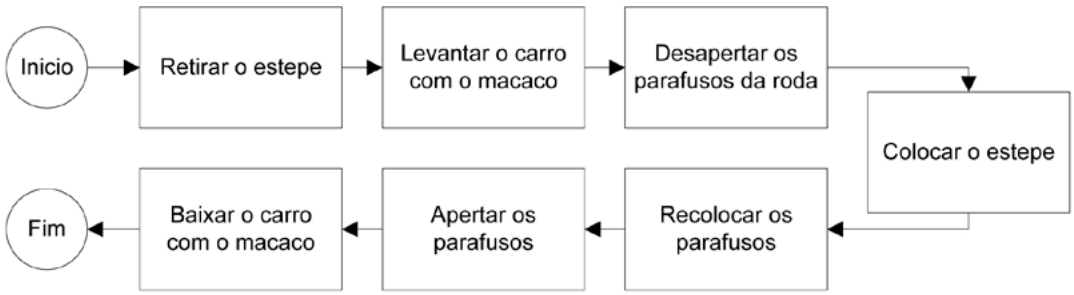
Constantemente, você utiliza sequências lógicas, criando algoritmos ou utilizando algoritmos já conhecidos e nem sabe, por exemplo: a regra de 3 é um algoritmo matemático muito conhecido e utilizado. O mesmo acontece quando você pensa em sair para o trabalho. Na sua cabeça um algoritmo é criado, ou seja, sequências lógicas para que você possa sair.

Exemplo de algoritmo utilizando na linguagem natural:

Para iniciar vamos:

- Retirar o estepe.
- Levantar o carro com o macaco.
- Desapertar os parafusos da roda.
- Colocar o estepe.
- Recolocar os parafusos.
- Apertar os parafusos.
- Baixar o carro com o macaco.
- Finalizar.

Note que este algoritmo é simples e com poucos passos. Você poderá aperfeiçoá-lo com mais passos através de exercícios. Tente melhorar este algoritmo refinando os passos.
O mesmo algoritmo pode ser representado de forma gráfica, é o que chamamos de fluxograma. Vamos a um exemplo:



O fluxograma é normalmente mais simples de entender, inclusive para os usuários ou clientes, já que estes não possuem conhecimento em programação, mas eles conseguem entender os passos lógicos ao ver o fluxograma.

GLOSSÁRIO

Fluxograma: diagrama esquemático com imagens que representa uma sequência de operações.
Algoritmo: processo de resolução de problemas constituído de uma sequência lógica, ordenada e bem definida.

Recordando

Neste capítulo, você aprendeu o que é Lógica e o que é Lógica de Programação. Viu ainda o que é programar e o que é algoritmo. Aprendeu que o que é importante para a construção de algoritmos e o que é um fluxograma.

No próximo capítulo veremos como construir algoritmos e fluxogramas. Estude e pratique constantemente a lógica.

1.2 - EXERCÍCIOS PROPOSTOS

1) Refine o algoritmo de Troca de Pneus.

_____
_____
_____

2) Crie um algoritmo para calcular a média de um aluno, sendo que a média é igual a nota do primeiro bimestre mais a nota do segundo bimestre dividido por dois.

_____
_____
_____

2 - FASES DO DESENVOLVIMENTO DE UM PROGRAMA

AS FRASES DO DESENVOLVIMENTO DE SISTEMAS
Como o sistema foi descrito no Levantamento inicial
Como foi definida pelo Análise
Como foi especificado no projeto
Como foi implementado
Como foi corrigido pela Manutenção
O que o usuário queria!

2.1 - FASES CLÁSSICAS

No início do desenvolvimento de sistemas não havia métodos bem definidos, isto porque a maioria dos programas era pequeno e apenas efetuavam cálculos. Com o crescimento dos sistemas entre as décadas de 60 e 80, novos modelos tiveram de ser utilizados, apenas desenvolver não era mais a única tarefa, deveriam ser feitas tarefas para analisar as necessidades, saber o que o cliente queria, surgindo então o modelo clássico para o desenvolvimento.

A figura a seguir apresenta o modelo cascata, mas que podemos denominar também como modelo circular, já que ao chegar à última fase, a fase de manutenção, retornamos à primeira fase.

Vamos entender cada etapa deste modelo.

- **Estudo de viabilidade**

Identificação de deficiências atuais e de objetivos a serem atingidos, para a elaboração de um cronograma de atividades e de uma relação Custos X Benefícios, relativas às soluções a serem desenvolvidas.

- **Análise**

Transformação do ambiente e das necessidades do usuário, impondo limites, desenvolvendo um modelo que possa descrever o que deve ser feito.

- **Projeto**

Desenho do projeto utilizando alguma metodologia: Análise estruturada e/ou essencial, Análise orientada a objetos, etc. É a descrição de como realizar logicamente e fisicamente o que será desenvolvido.

- **Implementação**

Nas fases anteriores, de planejamento, não envolvia linguagem de programação. No momento da implementação é a hora de codificar seu programa, envolve ainda a integração dos módulos.

- **Implantação**

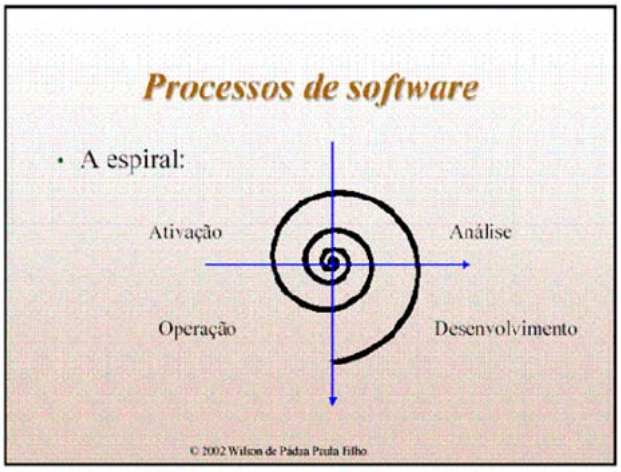
Compreende a conversão do arquivo, treinamento, instalação, etc.

- **Manutenção**

Acompanhamento preventivo e corretivo do sistema.

Estas fases são comuns também em outros modelos, como veremos adiante.

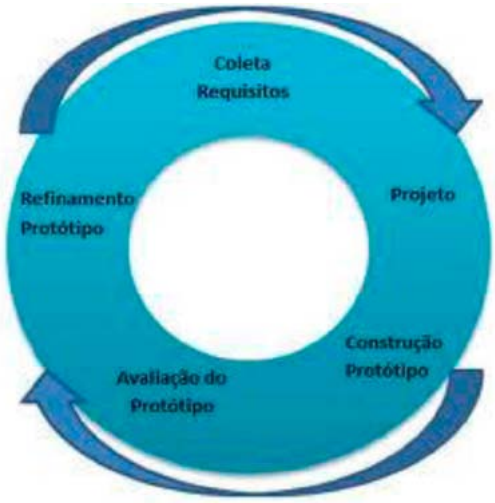
2.2 - MODELO ESPIRAL



O modelo expiral é muito semelhante ao modelo cascata, mas com um ponto forte, ele faz uma análise mais resumida, começa o desenvolvimento, segue as fases semelhantes à fase do modelo cascata, mas ao invés de planejar o sistema inteiro, planeja parte do sistema, desenvolve parte do sistema e vai dando andamento.

Note que o formato é um expiral que vai crescendo, ou seja, as fases vão atendendo mais e mais partes do sistema.

2.3 - MODELO PROTOTIPAGEM



O modelo por prototipagem trabalha de uma forma mais pragmática, apresentando um modelo ou protótipo para o cliente. Ao avaliar o protótipo, o cliente já sabe como o sistema deve ficar e o desenvolvedor vai atuar no refinamento do protótipo para tornar um produto real e completo.

O modelo de prototipação é prático, pois o cliente consegue ver parte do produto, mesmo que este ainda não esteja pronto.

2.4 - MODELO ÁGIL

Atualmente a metodologia ágil tem crescido rapidamente entre os modelos de desenvolvimento, visto que ele é funcional e ágil.

O que acontece no modelo ágil scrum, por exemplo, é que o desenvolvimento trabalha por fases. A cada fase, que dura de duas a quatro semanas, o cliente já deve receber partes do produto pronto e funcionando. Esta fase é denominada Sprint.

Este modelo ainda gera testes diariamente, pois é feito um ciclo de melhoria contínua, fazendo com que o produto chege ao cliente com pouca ou nenhuma falha.

Com os requisitos, os desenvolvedores separam as atividades por blocos que podem ser concluídas em ciclos de duas a quatro semanas a depender da empresa, como pode ser visto na figura anterior. Logo, o cliente recebe estas atividades prontas a cada ciclo.

Fica a dica

Existem ainda outros métodos. Faça uma pesquisa sobre a metodologia RUP, atualmente pertencente a IBM e sobre o Scrum que é o método ágil citado anteriormente.

Recordando

Neste capítulo, você aprendeu sobre os modelos de desenvolvimento e os seus ciclos. Viu ainda os modelos Clássicos, Expiral, Protótipo e Ágil. Aprendeu a importância das fases do desenvolvimento e para que servem.

No próximo capítulo veremos como construir algoritmos.

2.5 - EXERCÍCIOS PROPOSTOS

1) No modelo clássico/cascata qual a função da fase de Implementação?

_____
_____
_____

2) Qual a principal diferença entre o Modelo Clássico e o Modelo Ágil?

_____
_____
_____

3 – ALGORITMO

3.1 - ALGORITMO E FLUXOGRAMA

Fluxograma é a representação gráfica do algoritmo, sendo constituído de blocos funcionais que mostram o fluxo dos dados.

Para criar um algoritmo é necessário descrever a sequência de instruções, de maneira simples e muito objetiva. Para isso, é importante seguir algumas técnicas:

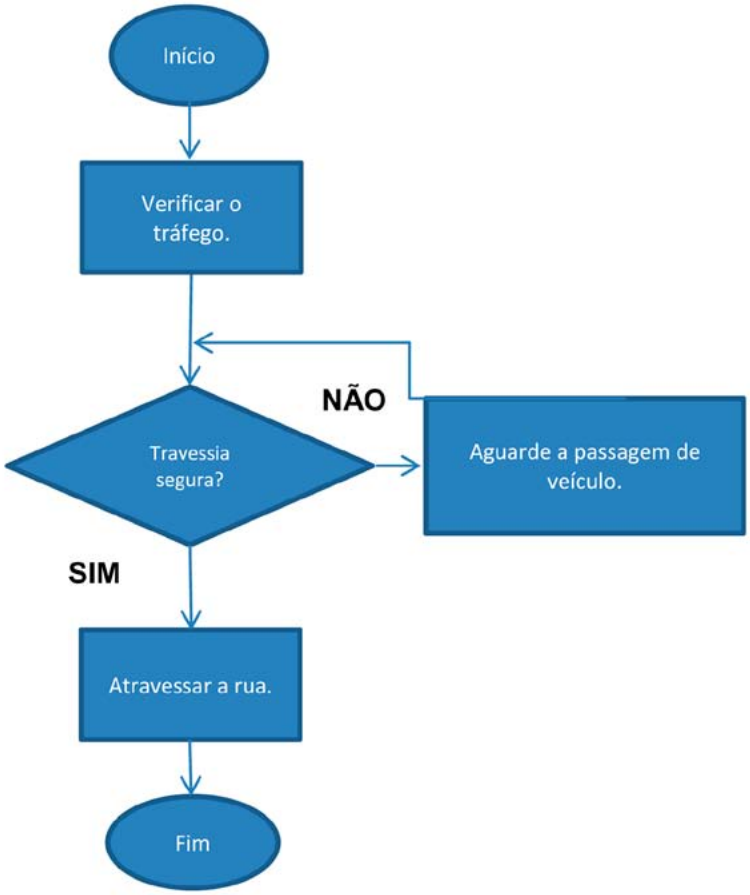
- Utilizar um verbo por frase.
- Imaginar que você está desenvolvendo um algoritmo para alguém que não conhece o que está sendo ensinado.
- Usar frases curtas e simples.
- Ser objetivo.
- Procurar usar palavras que não tenham sentido duplo.

Vamos a um exemplo:

Algoritmo: Atravessar a rua

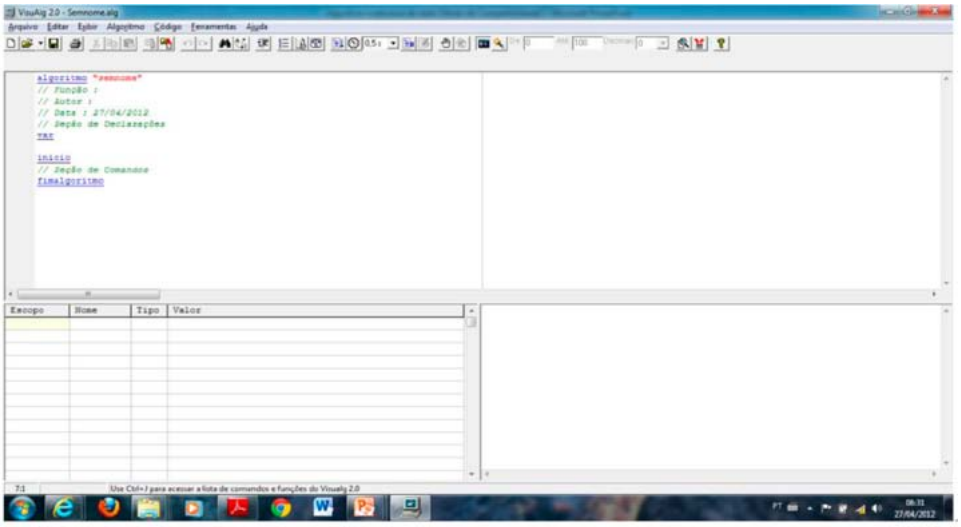
1. Verificar o tráfego.
2. Se a travessia é segura.
3. Então atravessa a rua.
4. Caso contrário, aguarde e tente novamente.

A representação gráfica deste algoritmo, ou seja, a representação do fluxograma deve ficar da seguinte forma:



Para a construção dos algoritmos, vamos utilizar o programa VisuALG, que está disponível gratuitamente em http://www.apoioinformatica.inf.br/programas. Este programa será utilizado para testar alguns algoritmos que vamos criar no decorrer do nosso curso.

Faça o download e instale. Ao abrir o VisuALG você verá a tela a seguir.

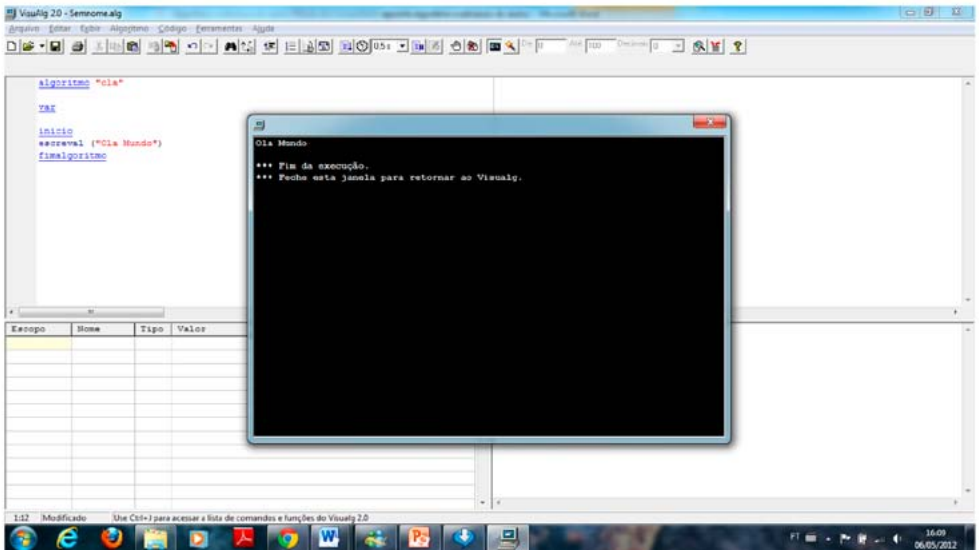


Vamos contruir nosso primeiro algoritmo.

Com base no fluxograma, vamos criar um algoritmo. Este é o algoritmo clássico "Olá mundo". Agora vamos codificar utilizando português estruturado ou portugol, dentro do VisuALG.

```
algoritmo "Ola"
var
inicio
escreval ("Ola Mundo")
fimalgoritmo
```

Clique em algoritmo **-->** Executar ou pressione F9 que o seu código será executado. O resultado será:



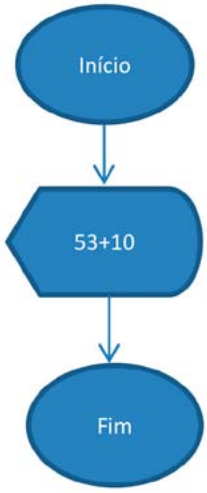
Para montar um algoritmo precisamos, primeiramente, dividir o problema em Entrada **-->** Processamento **-->** Saída. No caso do exemplo anterior, não temos entrada, temos apenas o processamento e a saída. O comando que possibilita a saída de dados foi o escreval("). Este comando indica ao VisuALG que é para exibir na tela alguma informação e pular uma linha. No nosso caso, queremos exibir um texto. Para exibir textos é necessária a utilização de aspas.

Exemplo: Escrever Meu nome é Fabio

Escreval ("Meu nome é Fabio")

Vamos agora fazer um novo algoritmo com o processamento de cálculo, conforme o

fluxograma a seguir.




```
algoritmo "soma"
var
inicio
escreval (53+10)
fimalgoritmo
```

Ao executar, o programa vai efetuar o cálculo de 53 mais 10, resultando em 63. Note que, ao fazer o cálculo, não foi necessário utilizar as aspas que utilizamos quando queremos escrever apenas texto na tela.

Recordando

Neste capítulo, você aprendeu a criar fluxogramas e algoritmos. Aprendeu também como testar os algoritmos com o VisuALG.

No próximo capítulo veremos como aperfeiçoar nossos algoritmos e como utilizar recursos.

3.2 - EXERCÍCIOS PROPOSTOS

1) Crie um algoritmo que faça a multiplicação de 4 números.

_____
_____
_____

2) Crie um algoritmo que exiba a média entre os números 8, 4, 6 e 9.

_____
_____
_____

4 – VARIÁVEIS

4.1 - ALGORITMO E VARIÁVEIS

"Um algoritmo é uma receita para um processo computacional e consiste de uma série de operações primitivas, interconectadas devidamente, sobre um conjunto de objetos. Os objetos manipulados por essas receitas são as variáveis."

A variável é uma representação simbólica de um endereço de memória, ou seja, é um espaço de memória com um apelido.
Tipos de dados:

- **Alfanumérico ou String:** armazena textos.
- **Inteiro:** armazena números inteiros.
- **Real ou ponto flutuante:** armazena números reais.
- **Lógico ou booleano:** verdadeiro e falso.

Importante!

Sempre que criamos uma variável estamos reservando um espaço na memória para o nosso programa efetuar o armazenamento de dados. Ao criar as variáveis, prefira nomes sugestivos, assim fica mais fácil de dar manutenção posteriormente. É mais fácil lembrar que a variável CPF armazena o cpf do que a variável abc123.

As variáveis devem ser declaradas antes de serem utilizadas. A declaração é o momento em que falamos o nome da variável e o tipo que ela vai armazenar.

Declarando váriáveis:

- **Nome da variável:** tipo

Exemplo:
media: real
Declarada a váriável média do tipo real (número real).
Vamos exercitar o uso de variáveis, melhorando nosso algoritmo de soma.

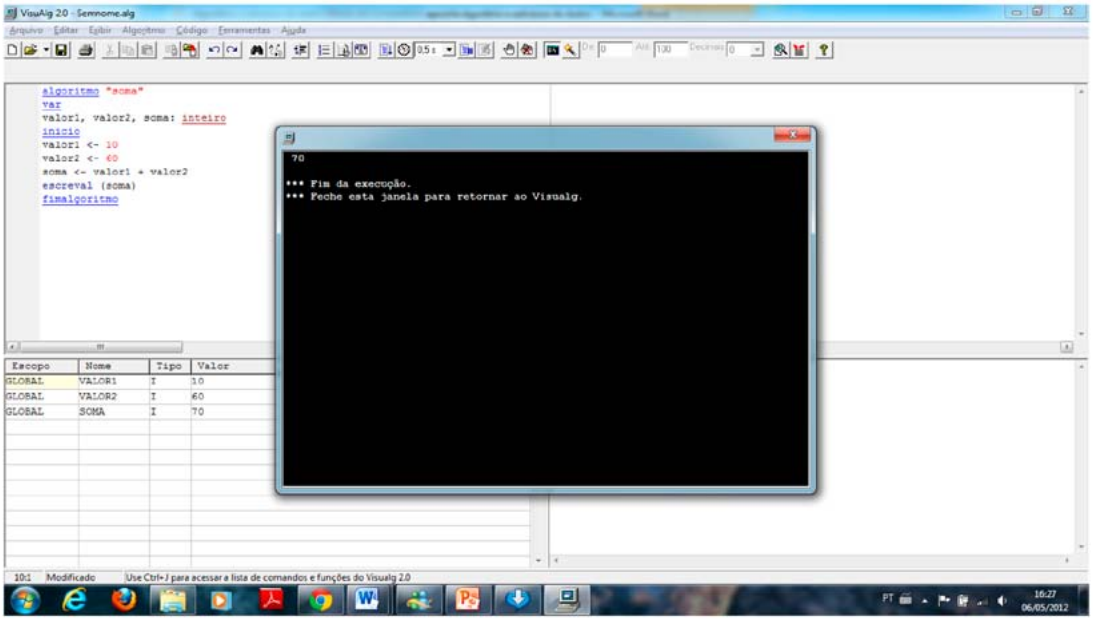
```
algoritmo "soma"
var
valor1, valor2, soma: inteiro
inicio
valor1 <- 10
valor2 <- 60
soma <- valor1 + valor2
escreval (soma)
fimalgoritmo
```

O sinal <- faz com que a variável receba um valor.

No exemplo foram criadas três variáveis, sendo valor1, valor2 e soma, todas do tipo inteiro.

Posteriormente, inserimos valores nas variáveis, valor1 colocamos o valor 10, valor2 foi colocado o valor 60 e para a variável soma foi feita a soma de valor1 com valor2.

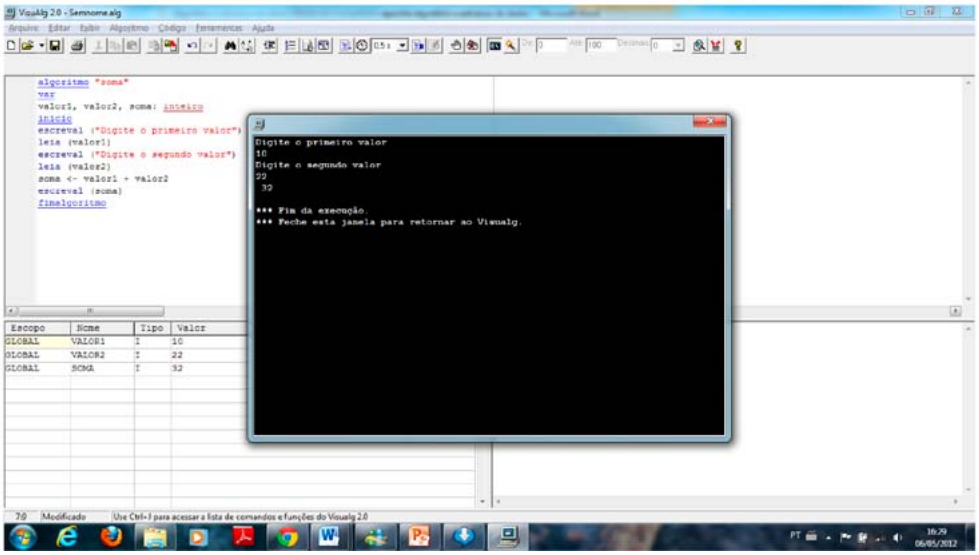
O comando escreval (soma) vai exibir na tela o resultado desta soma.



Com a variável podemos alterar os valores com mais facilidade, podemos ainda interagir com o usuário. Imagine que ao invés de deixar o valor fixo, agora vamos fazer uma entrada de dados vindo do usuário. Para isso, utilizaremos o comando leia. Vamos melhorar então nosso algoritmo de soma.

```
algoritmo “soma”
var
valor1, valor2, soma: inteiro
inicio
escreval (“Digite o primeiro valor”)
leia (valor1)
escreval (“Digite o segundo valor”)
leia (valor2)
soma <- valor1 + valor2
escreval (soma)
fimalgoritmo
```

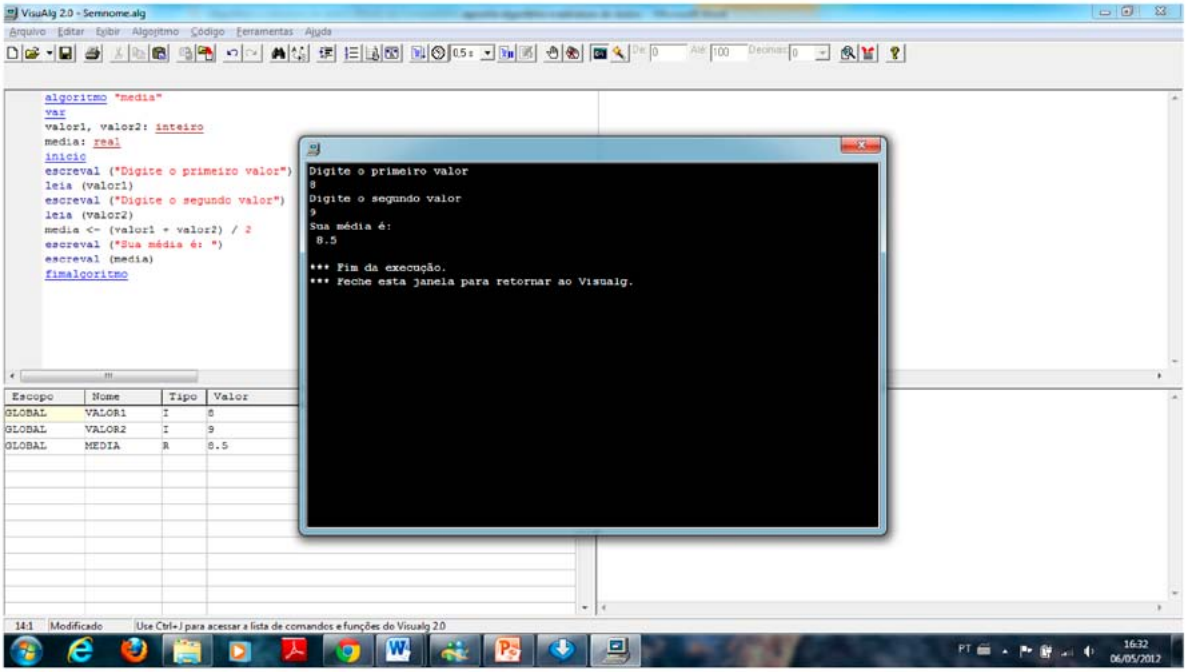
O resultado da execução é que nosso programa solicita do usuário um valor e este é armazenado na variável. Na figura a seguir temos a execução do programa.



Vamos agora fazer um programa de cálculo de média, semelhante ao que foi solicitado no exercício do capítulo anterior, mas com a diferença de que este interage com o usuário.

```
algoritmo “media”
var
valor1, valor2: inteiro
media: real
inicio
escreval (“Digite o primeiro valor”)
leia (valor1)
escreval (“Digite o segundo valor”)
leia (valor2)
media <- (valor1 + valor2)/2
escreval (“Sua media é: “)
escreval (media)
fimalgoritmo
```

Note que ao utilizarmos a divisão tivemos de colocar a soma entre parênteses, pois na ordem matemática, primeiro seria realizada a divisão depois a soma. Outra coisa que devemos notar é que a variável média está utilizando o tipo real, pois para fazer divisão não é permitida a utilização de números inteiros. O resultado da execução é a solicitação das notas e a exibição da média do aluno, conforme a figura a seguir.



Recordando

Neste capítulo você aprendeu a criar algoritmos utilizando variáveis. Viu ainda como criar algoritmos mais avançados que interajem com o usuário e entendeu os tipos de variáveis. Ainda colocou em prática o conhecimento realizando os testes no VisuALG.

4.2 - EXERCÍCIO PROPOSTO

1) Crie um algoritmo que faça a multiplicação de 4 números, mas agora sendo solicitado os dados do usuário.

_____

_____

_____

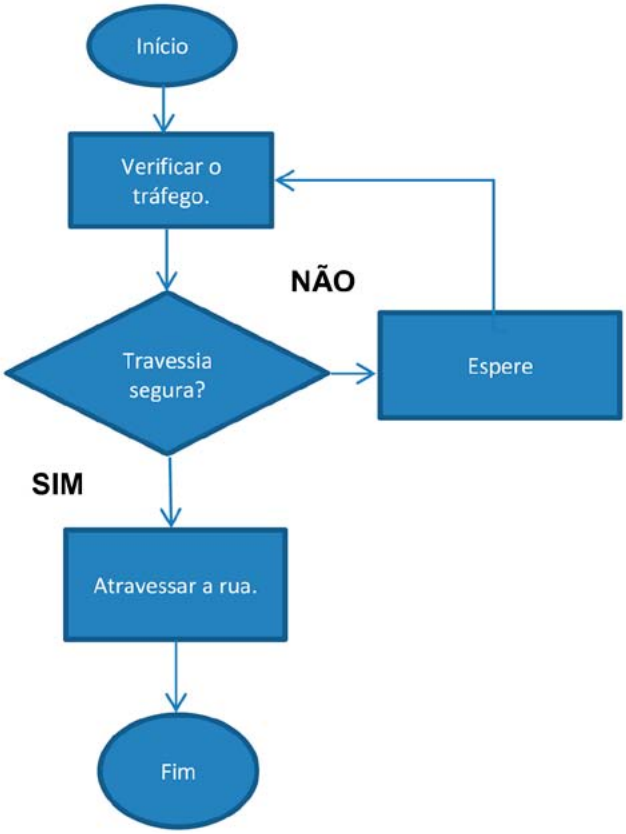
5 - ESTRUTURA DE DECISÃO

5.1 – CONDICIONAL

O processo condicional também é um processo de decisão ou escolha de alternativas. Utilizamos diariamente o sistema condicional, como por exemplo:

- Se chover então vou de carro, senão vou de bicicleta.

A condição para a comparação é a chuva e utilizamos o se para fazer a análise. Outro exemplo é quando estamos aguardando para atravessar a rua, olhamos se vem carro, se a travessia for segura nós atravessamos, senão ficamos aguardando. Em fluxograma a condicional é feita utilizando um losangulo como no exemplo a seguir.



Para tomarmos decisão, normalmente analisamos alguns pontos para verificar se é verdadeiro ou falso. No caso da chuva eu posso abrir a janela e ver se está chovendo ou como está o tempo. Com isso, estou fazendo um teste de condicional.

O teste pode analisar dois pontos. Vamos a um exemplo do aluno: se o aluno tiver nota acima de 6 e faltas abaixo de 25% ele está aprovado, caso contrário ele está reprovado, para isso, como devemos proceder nos testes?

Para simplificar o nosso teste, vamos utilizar o teste de mesa ou tabela-verdade.
A tabela-verdade é utilizada para validar nossa condicional. Veja a seguir.

SE	Media > 7	e	Falta < 20	então	Aluno aprovado
	V		V		V
	V		F		F
	F		V		F
	F		F		F

Com a utilização do e, as duas condições têm de ser verdadeiras para que o aluno seja aprovado, como pode ser notado na tabela anterior. Outra forma é a utilização do ou. Imagine que se o aluno tirar nota acima de 8 ou tiver menos de 20 faltas ele será aprovado. Neste caso, qualquer uma das afirmativas sendo verdadeira, o aluno é aprovado. Veja a tabela a seguir.

SE	Media > 7	ou	Falta < 20	então	Aluno aprovado
	V		V		V
	V		F		V
	F		V		V
	F		F		F

Note que a tabela facilita a visualização da condicional e dos testes, possibilitando verificar se a estrutura que foi criada está realmente funcional e clara.

Recordando

Neste capítulo, você aprendeu o que é condicional e o que é tabela-verdade. Aprendeu ainda como utilizar a tabela-verdade para fazer o teste lógico de sua condicional.

5.2 - EXERCÍCIO PROPOSTO

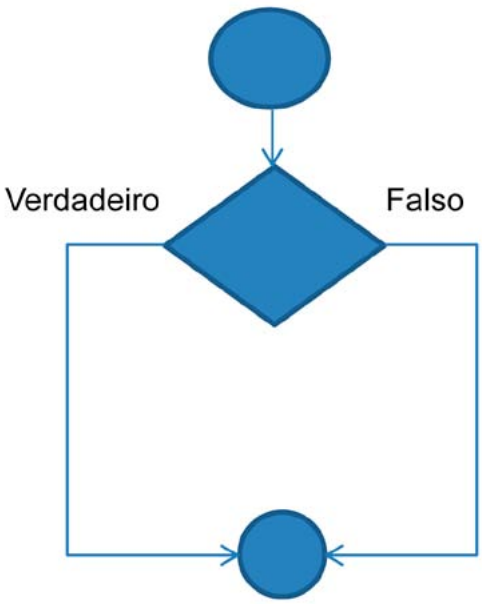
1) Com base no que você já aprendeu, faça um algoritmo utilizando condicional para dizer se o aluno está aprovado ou reprovado.

_____
_____
_____

6 - ALGORITMO DE ESTRUTURA DE DECISÃO

6.1 – CONDICIONAL

Funcionamento do SE (Condicional)
Se <condição> então
Expressão
Senão
Expressão
Fimse
Linguagem C, C++ e Java
If(condição){
Expressão
} else {
Expressão
}
Representação gráfica no fluxograma de condicional.



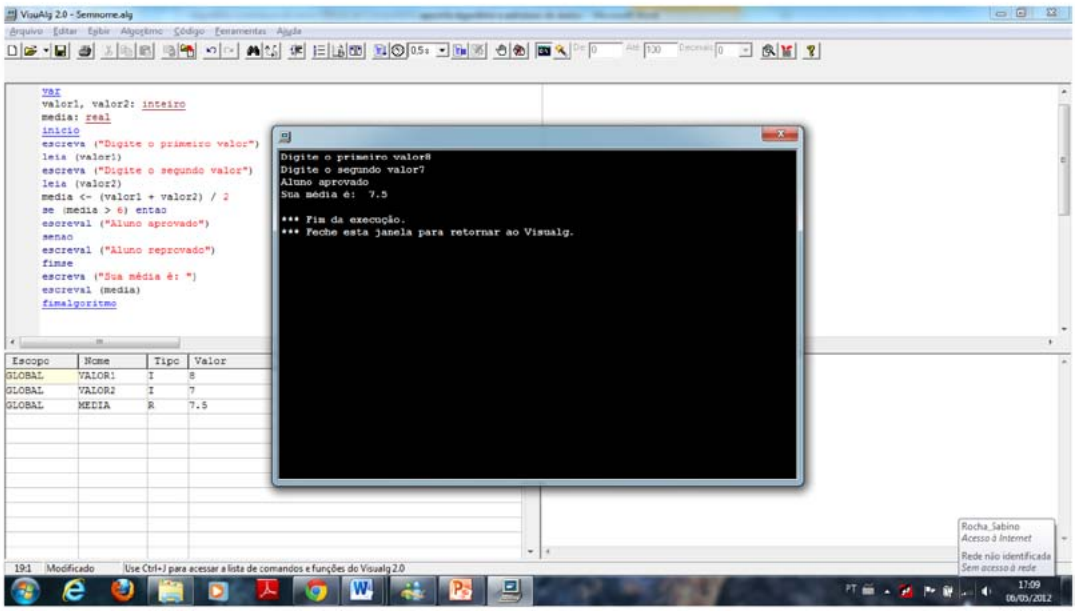
Vamos colocar em prática o uso da condicional em nosso programa de média. Imagine o seguinte problema:

Nosso programa já faz o cálculo de média, mas eu quero saber se o aluno está aprovado ou reprovado. Para isso tenho de comparar a nota, caso seja maior que 6 ele passou, caso seja menor que 6 o aluno está reprovado. Crie no VisuALG o seguinte algoritmo.
algoritmo "media"

var
valor1, valor2: inteiro
media: real
inicio
escreva ("Digite o primeiro valor")
leia (valor1)
escreva ("Digite o segundo valor")
leia (valor2)
media <- (valor1 + valor2)/2

```
se (media > 6) entao
escreval (“Aluno aprovado”)
senao
escreval (“Aluno reprovado”)
fimse
escreva (“Sua media é: “)
escreval (media)
fimalgoritmo
```

O resultado da execução do programa será:


Digite o primeiro valor8
Digite o segundo valor7
Aluno aprovado
Sua média é: 7.5
*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.

O se não é a única estrutura condicional existente. Temos um caso que é outra estrutura de decisão mais simples e permite diversas saídas.

O formato do caso é:

```
escolha <variável>
caso <condição>
   <Tarefa a ser realizada>
caso <condição>
   <Tarefa a ser realizada>
outrocaso
    <Tarefa a ser realizada>
fimescolha
```

Simples e direto, em linguagens como C, C++ e Java a codificação é:

```
switch(opc)
case 1:
   <Tarefa a ser realizada>
break;
case 2:
   <Tarefa a ser realizada>
break;
default:
break;
```

A base é a mesma, independente da linguagem.

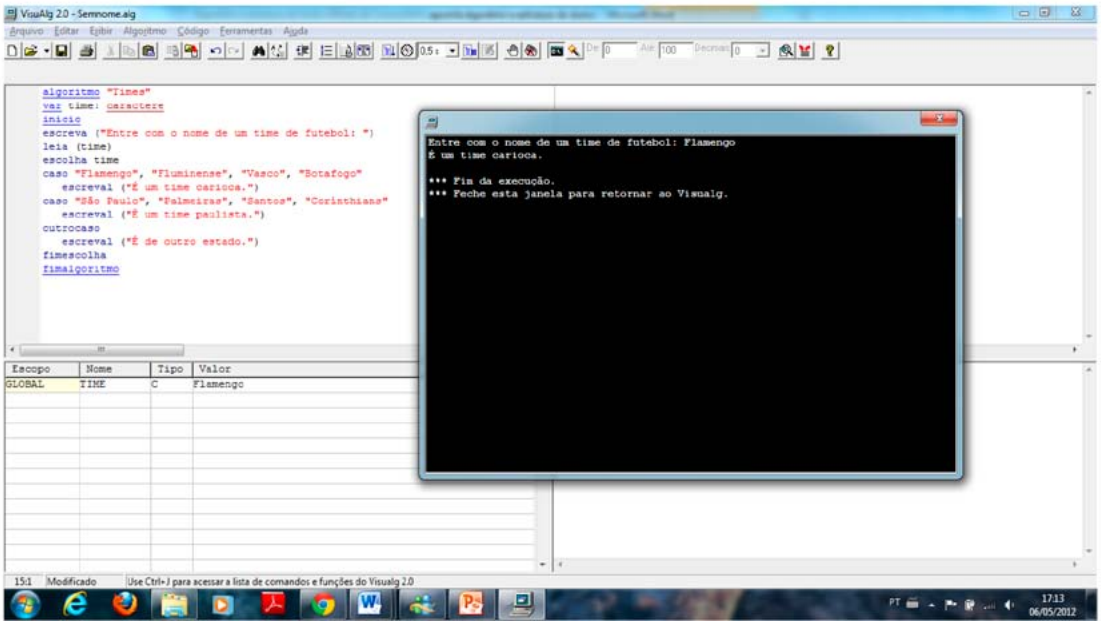
Vamos utilizar o caso para interagir com o usuário, solicitando informações e retornando

resultados.

O algoritmo a seguir pergunta o time que a pessoa torce. Ele compara se é de São Paulo ou do Rio de Janeiro, caso não seja nem de um lugar nem de outro ele diz que é de outro estado.

```
algoritmo “times”
var time: caractere
inicio
escreva (“Entre com o nome de um time de futebol: “)
leia (time)
escolha time
caso “Flamengo”, “Fluminense”, “Vasco”, “Botafogo”
    escreval (“É um time carioca.”)
caso “São Paulo”, “Palmeiras”, “Santos”, “Corinthians”
    escreval (“É um time paulista.”)
outrocaso
    escreval (“É de outro estado.”)
fimescolha
fimalgoritmo
```

O resultado da codificação será:



Recordando

Neste capítulo, você colocou em prática a utilização da Estrutura de Decisão. Apredeu ainda a Estrutura de Decisão caso, que em diversos momentos pode tornar o processo de desenvolvimento mais simples do que o se.

6.2 - EXERCÍCIO PROPOSTO

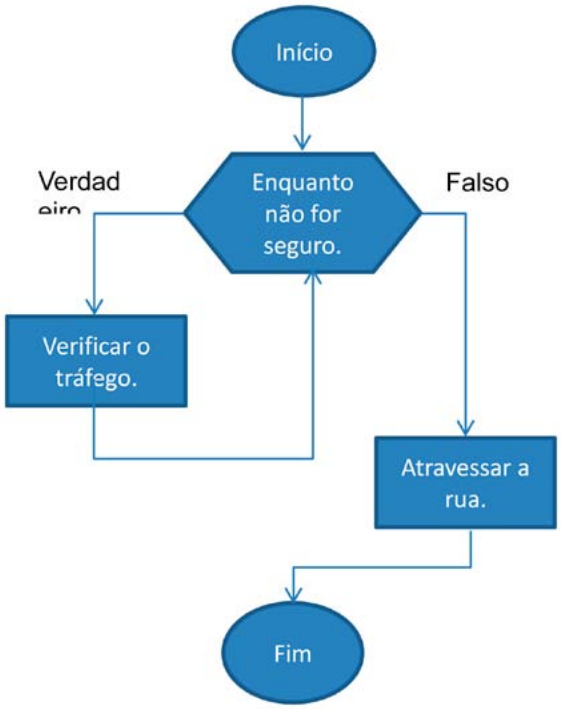
1) Utilize o exemplo que criamos de Time de Futebol e complete fazendo uma comparação com os times do seu estado.

_____
_____
_____

7 - ESTRUTURA DE REPETIÇÃO

A estrutura de repetição é utilizada quando desejamos que um conjunto de instruções seja executado em um número definido ou indefinido de vezes.

No fluxograma pode ser representada utilizando um hexágono como no modelo a seguir:



Informações!

"Os laços de repetição também são conhecidos por sua tradução em inglês: loops ou looping. Ganham esse nome por lembrarem uma execução finita em círculos, que depois seguem o seu curso normal."

Os laços de repetição são utilizados quando desejamos que um determinado conjunto de instruções seja executado em um número finito ou infinito de vezes.
Como exemplo disso temos:

O professor passa como castigo para o aluno que está de recuperação, que ele escreva 10 vezes no quadro. Então, temos aí a contagem de vezes.

Escreva 10 vezes o texto xyz.

Agora, um problema real! Você é o responsável pelo desenvolvimento de um sistema de cadastros de alunos. O problema é que serão cadastrados 50 alunos por turma e são 5 turmas. Imagine o tamanho do problema! Se você for fazer com o que aprendemos até o capítulo anterior, teria de criar 50 variáveis para cada turma e ainda fazer um código imenso para isso. Para este problema, a estrutura de decisão vai servir, ou seja, facilitar o trabalho repetitivo.

No fluxograma vimos a utilização do Enquanto, que é um processo comum que utilizamos diariamente, como no exemplo de atravessar a rua. Enquanto o sinal não estiver verde eu espero.

Recordando

Neste capítulo, você aprendeu o que é uma Estrutura de Repetição e viu como utilizá-la.

7.1 - EXERCÍCIO PROPOSTO

1) Com base no fluxograma apresentado, crie um fluxo para fazer uma ligação aguardando ser atendido.

_____

_____

_____

8 - ALGORITMO DE ESTRUTURA DE REPETIÇÃO

8.1 - ESTRUTURA DE REPETIÇÃO

A primeira estrutura de repetição que veremos é o Enquanto (while). Este verifica a condição e enquanto esta não for verdadeira, continua executando o código.
Funcionamento:
enquanto <condição> faca
Código a ser executado
fimenquanto

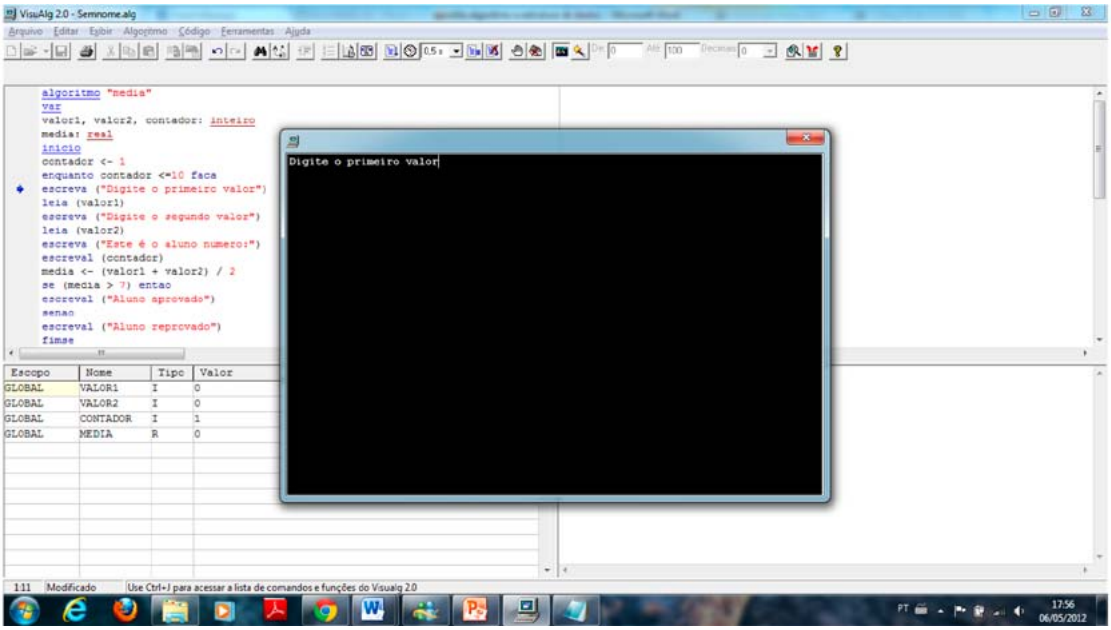
Vamos codificar no VisuALG.

```
algoritmo "Números de 1 a 10 (com enquanto...faca)"
var j: inteiro
inicio
j < - 1
enquanto j < = 10 faca
   escreva (j)
   j < - j + 1
fimenquanto
fimalgoritmo
```

Neste caso, o programa exibe informações enquanto o j for menor que 10, e enquanto está contando, ele exibe o valor na tela.

Agora imagine nosso algoritmo de média. Vamos fazer a média de 10 alunos.

```
algoritmo "media"
var
valor1, valor2, contador: inteiro
media: real
inicio
contador <- 1
enquanto contador <=10 faca
escreva ("Digite o primeiro valor")
leia (valor1)
escreva ("Digite o segundo valor")
leia (valor2)
escreva ("Este é o aluno numero:")
escreval (contador)
media <- (valor1 + valor2)/2
se (media > 7) entao
escreval ("Aluno aprovado")
senao
escreval ("Aluno reprovado")
fimse
escreva ("Sua media é: ")
escreval (media)
contador <- contador +1
fimenquanto
fimalgoritmo
```



Existe ainda outro formato de repetição, como vimos no capítulo anterior. O Para ou For é utilizado para uma quantidade finita de execuções.

Exemplo:

Para alunos de 1 a 20 faça.

Neste caso, estou dizendo que é para rodar de 1 até 20, ou seja, 20 repetições. Funciona semelhante ao enquanto, com a diferença do formado de comparação.

Vamos então à prática.

Vamos utilizar o Para e exibir números de 1 a 10, como fizemos com o Enquanto.

```
algoritmo “Números de 1 a 10”
var j: inteiro
inicio
para j de 1 ate 10 faca
   escreva (j)
fimpara
fimalgoritmo
```

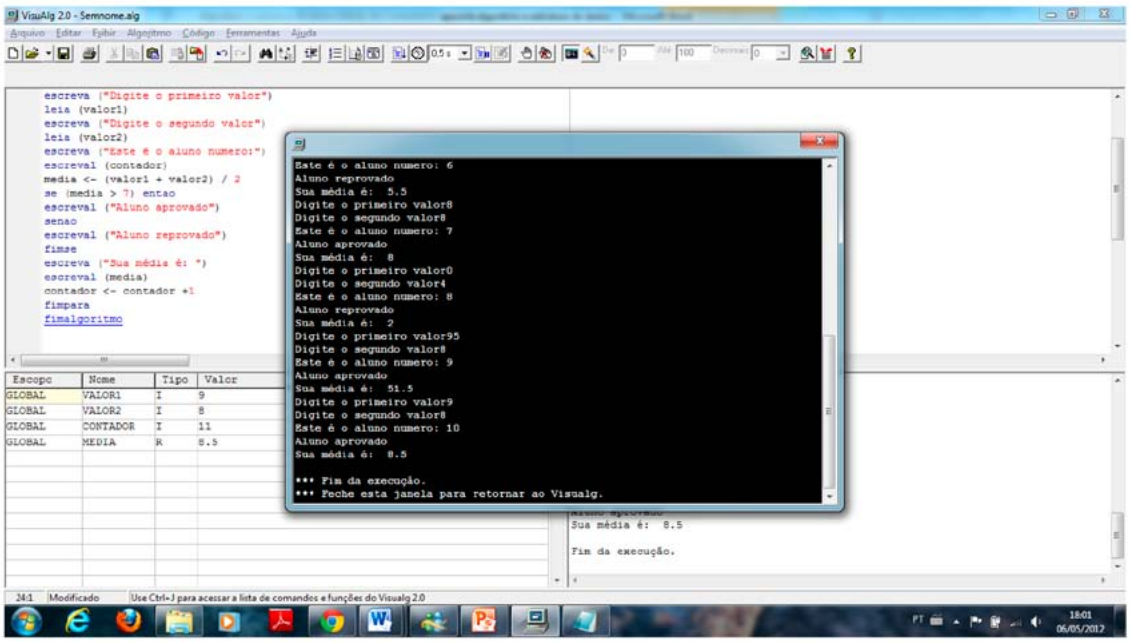
Note que apesar de diferente, o resultado é o mesmo.

Vamos agora utilizar o Para, para fazer o nosso programa média, da mesma forma que utilizamos o Enquanto.

```
algoritmo “media”
var
valor1, valor2, contador: inteiro
media: real
inicio
para contador de 1 ate 10 faca
escreva (“Digite o primeiro valor”)
leia (valor1)
escreva (“Digite o segundo valor”)
```

```
leia (valor2)
escreva (“Este é o aluno numero:”)
escreval (contador)
media <- (valor1 + valor2) / 2
se (media > 7) entao
escreval (“Aluno aprovado”)
senao
escreval (“Aluno reprovado”)
fimse
escreva (“Sua media é: “)
escreval (media)
contador <- contador +1
fimpara
fimalgoritmo
```

E o resultado será a execução 10 vezes da estrutura de média, conforme pode ser visto na figura a seguir:



Lembrete!

o laço de repetição termina quando a condição se tornar falsa.

Recordando

Neste capítulo, você aprendeu como utilizar as estruturas de repetição, Enquanto e Para, em seus algoritmos. Pratique constantemente para saber a hora de utilizar cada uma das estruturas.

8.2 - EXERCÍCIO PROPOSTO

1) Agora que você já conhece as estruturas, pratique criando um algoritmo que faça cadastro em uma agenda de 10 pessoas.

9 - ESTRUTURA DE DADOS: VETOR

9.1 - VARIÁVEIS COMPOSTAS: VETOR

A variável composta ou vetor, utilizado na matemática como matriz, é uma variável com diversos pontos. A exemplo temos as notas dos alunos, e como as notas são do mesmo aluno, ao invés de ter nota1, nota2, nota3, nota4, poderíamos ter apenas uma variável nota, só que com 4 posições.

O vetor funciona como uma variável com subvariáveis, desta forma, quando indicamos a criação do vetor com 4 posições como no exemplo anterior, o computador separa 4 posições de memória para a utilização do vetor.

A declaração da variável do tipo vetor é semelhante à declaração da variável que aprendemos anteriormente, com a principal diferença de indicar que é um vetor de X posições, em que o X indica a quantidade de posições.

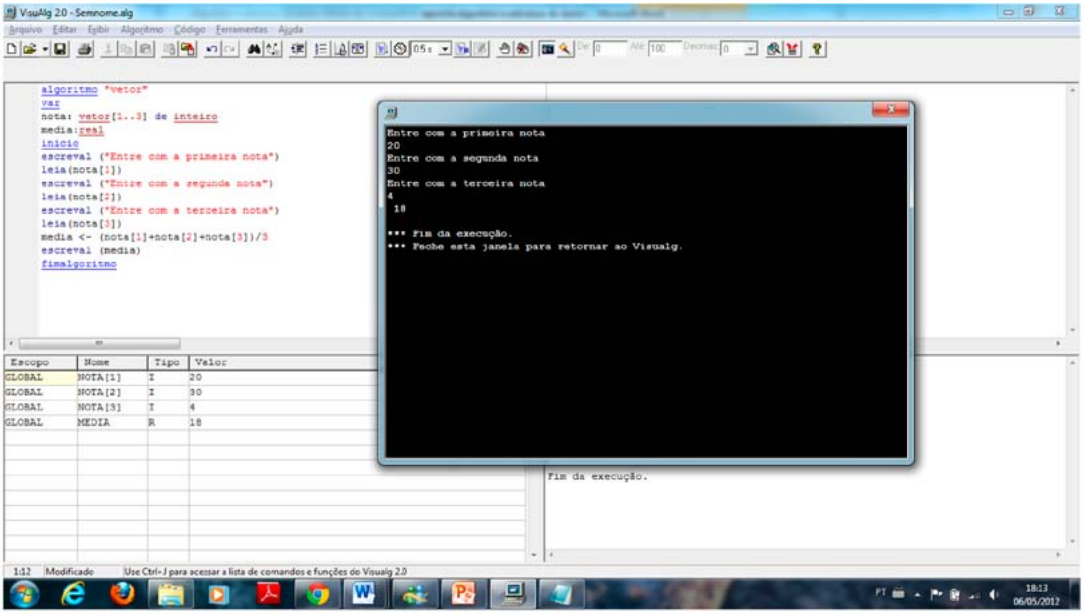
Imagine um vetor como um espaço com diversas colunas, sendo que cada coluna é uma posição dentro do vetor. Se ao criarmos o vetor falarmos que ele tem 5 posições, imagine 5 colunas.

Como declarar uma variável do tipo vetor:
Título: vetor[inicio..fim] de tipo
Exemplo:
Nota: vetor[1..4] de inteiro
Para utilizar o vetor, ou seja, para acessá-lo, basta utilizar o vetor semelhante a uma variável, mas indicando a posição (coluna) que você deseja a informação.
Vetor[posição]
Exemplo:
Nota[1]

Agora vamos à prática!

Vamos utilizar o vetor em nosso programa consagrado de média de alunos, só que ao invés de criarmos 2 ou 4 variáveis para armazenar as notas, utilizaremos um vetor.

```
algoritmo "vetor"
var
nota: vetor[1..3] de inteiro
media: real
inicio
escreval ("Entre com a primeira nota")
leia (nota[1])
escreval ("Entre com a segunda nota")
leia (nota[2])
escreval ("Entre com a terceira nota")
leia (nota[3])
media < - (nota[1]+nota[2]+nota[3])/3
escreval (media)
fimalgoritmo
```



Note no VisuALG que as variáveis foram criadas e indicadas na parte de baixo da janela.

Escopo	Nome	Tipo	Valor
GLOBAL	NOTA[1]	I	20
GLOBAL	NOTA[2]	I	30
GLOBAL	NOTA[3]	I	4
GLOBAL	MEDIA	R	18

Aqui temos as variáveis notas separadas como nota[1] coluna 1, nota[2] coluna 2 e assim por diante, com os seus respectivos valores.

Veja que a utilização do vetor reduz o problema de programação, pois possui menos variáveis a serem criadas de forma manual pelo programador, ou seja, quem vai criar as variáveis neste caso é o programa, o programador sabe que tem 4 posições com um único nome e pode utilizá-las, facilitando o desenvolvimento do sistema.

Informações!

O vetor é um tipo de variável com diversas posições. Tome cuidado para não criar posições que não serão utilizadas, desta forma seu algoritmo será mais eficiente.

Recordando

Neste capítulo, você aprendeu o que é um vetor e como utilizá-lo em seus algoritmos.

9.2 - EXERCÍCIO PROPOSTO

1) Agora que você já conhece o vetor, pratique criando um algoritmo que faça cadastro em uma agenda de 10 pessoas em vetor.

_____

_____

_____

10 - ESTRUTURA DE DADOS: MATRIZ

10.1 - VARIÁVEIS COMPOSTAS: MATRIZ

Vimos no capítulo anterior a variável composta do tipo vetor, que armazena dados sequencialmente em colunas. Vimos também o quanto ela pode ser útil para o nosso trabalho e para reduzir a quantidade de variáveis declaradas no início do sistema. Agora, se quisermos cadastrar as notas de 10 alunos? Continuamos tendo um problema, pois a quantidade de variáveis continua grande. Para este problema devemos utilizar outro tipo de variáveis compostas, a matriz.

A matriz também é uma variável composta semelhante ao vetor tratado no capítulo anterior, a principal diferença é que no vetor temos apenas uma dimensão, ou na verdade apenas colunas, e na matriz temos diversas dimensões, ou linhas e colunas.

A matriz cria uma solução mais prática para o problema de cadastro de notas dos 10 alunos, pois como temos linhas e colunas podemos ter dez linhas com quatro colunas, desta forma atingiremos o nosso objetivo.

O que temos é semelhante a um vetor, portanto temos uma variável que ocupa diversos espaços de memória, um espaço multidimensional. Então, quando criamos uma variável de duas linhas e quatro colunas, estamos indicando para o sistema separar oito posições de memória.

Para criar na prática uma matriz temos:

Nome: vetor[inicio linha..fim linha, inicio coluna..fim coluna] de tipo

Exemplo:

Nota: vetor[1..10,1..4] de inteiro
Acesso
Nome[linha,coluna]
Exemplo:
Nota[1,1]

Note que a codificação é semelhante ao vetor. Na verdade, a única diferença é que agora temos dois pontos, as linhas e depois as colunas.

O acesso também é idêntico, a principal diferença é a quantidade de informações necessárias para acessar a informação.

Vamos a um exemplo básico utilizando matriz:

```
algoritmo “semnome”
var
nota: vetor[1..10,1..3] de inteiro
media: real
inicio
escreval (“Entre com a primeira nota”)
leia (nota[1,1])
escreval (“Entre com a segunda nota”)
leia (nota[1,2])
escreval (“Entre com a terceira nota”)
leia (nota[1,3])
```

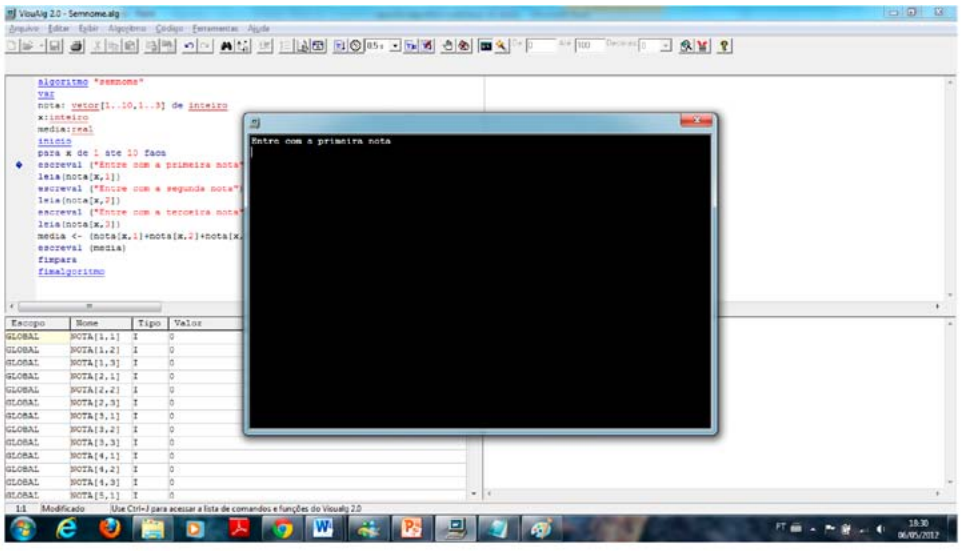
```
media < - (nota[1,1]+nota[1,2]+nota[1,3])/3
escreval (media)
fimalgoritmo
```

Este algoritimo criado apenas cadastra nota de um aluno, agora imagine se quiser cadastrar as notas dos 10 alunos, como proceder?

A solução já foi vista anteriormente. Vamos utilizar uma estrutura de repetição e aí efetuamos o cadastro.

```
algoritmo “semnome”
var
nota: vetor[1..10,1..3] de inteiro
x: inteiro
media: real
inicio
para x de 1 ate 10 faca
escreval (“Entre com a primeira nota”)
leia (nota[x,1])
escreval (“Entre com a segunda nota”)
leia (nota[x,2])
escreval (“Entre com a terceira nota”)
leia (nota[x,3])
media <- (nota[x,1]+nota[x,2]+nota[x,3])/3
escreval (media)
fimpara
fimalgoritmo
```

Veja que este algoritmo além de reduzir a quantidade de códigos, permite que façamos a média de 10 alunos, diferente dos algoritmos anteriores. Outra vantagem é que automaticamente o sistema vai do aluno 1 para o aluno 2. Veja na parte inferior que o sistema cadastra informações nas variáveis do seu sistema.



O exemplo pode ser alterado com facilidade para cadastrar nota de 20, 40 ou até 100 alunos sem problemas.

Informações!
A matriz é uma estrutura espetacular que facilita o trabalho do programador, mas cuidado para não criar matrizes com tamanhos desnecessários, só para evitar que o sistema mude, assim você evita o estouro de memória.

Recordando

Neste capítulo, você aprendeu o que é uma matriz e como utilizá-la em seus algoritmos.

10.2 - EXERCÍCIO PROPOSTO

1) Agora que você já conhece o vetor, pratique criando um algoritmo que faça cadastro em uma agenda de 10 pessoas em uma matriz com nome e e-mail.

_____

_____

_____

11 - FUNÇÃO

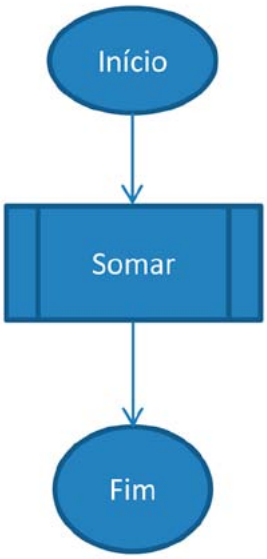
Até o momento vimos como criar estruturas de repetição, mas nosso programa possui um ciclo que vai do início ao fim. A função é uma das formas utilizadas para modularizar um programa, tornando-o mais fácil de entender e de efetuar manutenções.

Um problema normalmente pode ser dividido em problemas menores. Estes problemas menores serão solucionados e esta solução será dada por uma função.

Quando trabalhamos com desenvolvimento de sistemas, não conseguimos solucionar um grande problema, então para facilitar, dividimos o grande problema em diversos problemas menores, facilitando assim a busca por solução, só que até o momento nossa solução é contínua, sem divisão.

Agora imagine uma solução que ocupe centenas de linhas. Ao termos um problema em uma linha, quanto tempo leva para dar a manutenção?

E as alterações, divisão de tarefas em equipes, como vamos proceder? A solução é modularizar e a utilização de funções possibilita esta modularização. Em fluxograma, o exemplo fica da seguinte forma:



O símbolo indica um módulo, procedimento ou função com código adicional.
Na prática, a criação da função segue a seguinte sintax:

Função nome da função: tipo
Fimfunção
Vamos fazer o programa, seguindo nosso fluxograma de soma.

```
algoritmo "funcaosoma"
var
   n,m,res:inteiro
inicio
funcao soma: inteiro
var aux: inteiro
inicio
aux < - n + m
retorne aux
```

```
fimfuncao
n < - 4
m < - -9
res < - soma
escreva(res)
fimalgoritmo
```

Ao executar o programa, ele rodará, mas a função soma só sera executada quando a linha res < - soma chegar.

Informações!

A utilização da função facilita a manutenção, reduzindo assim problemas, já que podemos dar manutenção em uma função sem ter de alterar toda a codificação.

Recordando

Neste capítulo, você aprendeu o que é uma função e como modularizar seus algoritmos.

11.1 - EXERCÍCIO PROPOSTO

1) Faça um algoritmo com função para soma e subtração.

_____
_____
_____

12 - FUNÇÃO COM PASSAGEM DE PARÂMETRO

No capítulo anterior, vimos como trabalhar com função, modularizando nossos programas. A função por si só já facilita o entendimento do código, mas a criação de variáveis globais e a falta de entrada de dados direto na função complica a sua utilização. Para este problema, existe a passagem de valor por parâmetro.

Sintax:
Função
Nome (parâmetros): retorno
Inicio
Retorne retorno
Fimfuncao

A utilização de uma função com parâmetro deve ser feita com o nome seguido pelos parâmetros e os dados.

Nome (parâmetro1, parâmetro2).
Utilizando o mesmo modelo da função do capítulo anterior de soma, vamos criar uma função com passagem de parâmetros.

```
algoritmo "funcaoparametro"
var
   n,m,res:inteiro
inicio
funcao soma (x,y: inteiro): inteiro
inicio
retorne x + y
fimfuncao
n < - 4
m < - -9
res < - soma (n,m)
escreva (res)
fimalgoritmo
```

Note que ficou mais fácil de manipular e ao executar, a variável não apareceu como global porque ela roda apenas dentro da função.

Agora vamos utilizar a função com parâmetro e uma estrutura de decisão.

```
algoritmo "fparametro"

var
   n,m,res, opc: inteiro
inicio
funcao soma (x,y: inteiro): inteiro
inicio
retorne x + y
fimfuncao
funcao subtracao (x,y: inteiro): inteiro
inicio
retorne x - y
fimfuncao
escreva ("Digite 1 para soma ou 2 para subtração")
leia (opc)
escreva ("Digite o primeiro valor")
leia (n)
```

```
escreva (“Digite o seguindo valor”)
leia (m)
escolha (opc)
caso 1
    res < - soma (n,m)
caso 2
    res < - subtracao (n,m)
outrocaso
        escreva (“Não foi escolhido nem soma e  nem subtração”)
fimescolha
escreva(res)
fimalgoritmo
```

No algoritmo anterior, utilizamos a escolha para definir a utilização da função soma ou subtração. Verificamos também que caso não seja digitado um valor válido, o programa emite uma mensagem.

Recordando

Neste capítulo, você aprendeu o que é uma função com parâmetros e como modularizar seus algoritmos utilizando este recurso.

Saiba Mais!

www.fgrweb.com.br
www.portaldaprogramacao.com

12.1 - EXERCÍCIO PROPOSTO

1) Complete o segundo algoritmo deste capítulo com multiplicação e divisão.

13 - REGISTRO E ESTRUTURA

O que é um Registro ou Estrutura?

Um Registro ou Estrutura é um conjunto de informações logicamente relacionadas, sendo uma das principais estruturas de dados.

Um registro é composto por campos.

Exemplo de Registro/Estrutura:

Estrutura Aluno

Inicio

Nome: caractere
Idade: inteiro
Curso: caractere
Fimestrutura

Veja que no exemplo anterior temos uma estrutura denominada aluno. Esta estrutura contém diversas variáveis, que fazem parte da estrutura, sendo elas: nome, idade e curso. Desta forma, juntamos conteúdos semelhantes em torno de uma única estrutura de dados.

O acesso à estrutura é feito criando-se uma nova variável, só que ao invés de ser do tipo inteiro, caractere e etc., ela será do tipo da estrutura, podendo acessar suas subinformações.

Exemplo:

Aluno1 aluno
Alunol.nome < - "fabio"
A criação de um conjunto de registro ou estrutura pode ser possível, para isso, basta utilizar vetores.
Aluno1 vetor [1..100] de aluno
Aluno1[1].nome < - "fabio"

RECORDANDO

Neste capítulo, você aprendeu o que é uma estrutura, viu que ela serve para criar conjunto de informações relacionadas, facilitando a manutenção do sistema e o entendimento da lógica.

13.1 - EXERCÍCIO PROPOSTO

1) Cite um outro exemplo de registro que você poderia utilizar.

_____
_____
_____

14 - ARQUIVOS

Um arquivo é um conjunto de registros/dados armazenados em uma memória auxiliar. Os primeiros microcomputadores não possuíam disco rígido para armazenar os dados, eram utilizados disquetes. Atualmente, os arquivos podem estar em diversos dispositivos, como disquete, HD, Pen drive e etc.

O arquivo permite armazenar dados por um período de tempo indeterminado, diferente do que acontece com o armazenamento em memória Ram, que ao desligar o computador os dados são perdidos.

Os sistemas atualmente precisam lidar com arquivos, visto que é de grande importância o armazenamento das informações por períodos longos, e também caso ocorram erros, as informações devem estar armazenadas no computador.

Para trabalhar com arquivos devemos seguir uma sequência:

- Cria uma variável que simboliza o arquivo.
- Abre o arquivo, se o arquivo existir, se não existir cria um novo arquivo.
- Pega os dados e insere no arquivo.
- Fecha o arquivo.

Esta sequência simples demonstra os passos básicos necessários para manipular arquivos.

Um exemplo prático no VisualG:

```
algoritmo "lendo do arquivo"
arquivo "teste.txt"
var x,y: inteiro
inicio
para x de 1 ate 5 faca
    leia (y)
fimpara
fimalgoritmo
```

O arquivo é a função que abre ou cria um novo arquivo. Neste caso, o nome do arquivo é teste.txt.

Todos os dados são gravados no arquivo, automaticamente no VisualG.

Abaixo será criado um sistema de agenda utilizando a mesma estrutura:

```
algoritmo "lendo do arquivo"
arquivo "agenda.txt"
var
nome: caractere
fone, x, y: inteiro
inicio
escreval ("Quantos contatos deseja cadastrar?")
leia (y)
para x de 1 ate y faca
escreval ("Digite o nome")
leia (nome)
escreval ("Digite o fone")
leia (fone)
fimpara
```

fimalgoritmo

Fique ligado!

Com o arquivo é possível criar programas que armazenam informações diversas, então aproveite o exemplo da agenda inserindo informações de e-mail, data de aniversário e outras informações.

15 - ORDENAÇÃO

Ordenação é um dos fatores primordiais para o funcionamento de um programa que trate de dados, imagine ter uma relação de informação desordenada?! Para efetuar uma busca será necessário muito mais tempo para conseguir achar uma informação. Existem diversas formas de efetuar a busca, que é através dos Métodos Simples e dos Métodos Sofisticados.

15.1 - MÉTODOS SIMPLES

Os métodos simples facilitam a implementação por serem menos complexos. São indicados para:

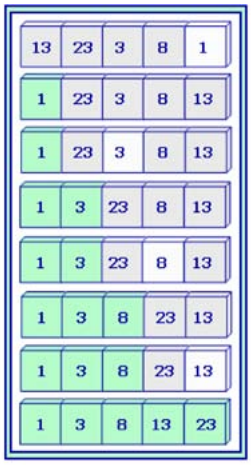
- Conjuntos pequenos de informações.
- Usam muitas comparações.
- Os códigos são pequenos.
- Códigos de fácil entendimento.
- Mais eficientes para pequenos arquivos.

Exemplos:

- *Selection Sort*
- *Insertion Sort*
- *Bubble Sort*

15.1.1 - SELECTION SORT

O princípio básico desse método é sempre buscar o menor dos valores restantes e levá-lo às posições iniciais (ordenação crescente).



A figura acima apresenta um exemplo de Selection Sort, que é indicado:

- Ideal para arquivos pequenos.
- Muito simples.
- Não melhora sua performance se o arquivo já estiver ordenado.
- Tem menos movimentos do que o Insertion Sort.

15.1.2 - INSERTION SORT

O princípio básico desse método é considerar o vetor como dois subvetores, um ordenado e o outro desordenado, buscando posicionar o elemento que se encontra no subvetor

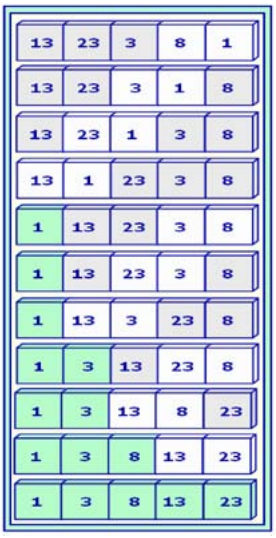
desordenado, no vetor ordenado.

Os principais benefícios deste método são:

- Só ordena quando necessário.
- Quando um elemento é inserido, todos os elementos maiores que ele são deslocados para a direita.
- É considerado o melhor dos três métodos estudados.

15.1.3 - BUBBLE SORT

O princípio básico desse método é trocar de posição toda vez que forem encontrados valores de posições adjacentes fora de ordem.



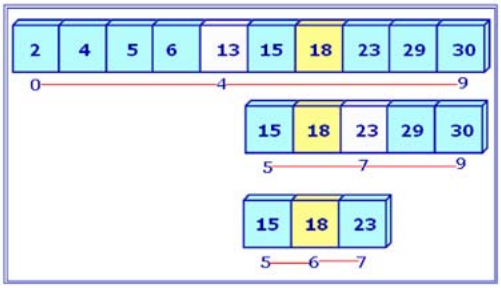
Como benefícios deste método temos:

- É o mais conhecido método.
- É muito simples.
- Muito lento.
- É considerado o pior dos três estudados.

16 - PESQUISA

A pesquisa é o ato de indagar, investigar, procurar com diligência. Os dois métodos principais de pesquisa é o Sequencial ou Linear e o Binário.

A Busca Sequencial/Linear: tipo de pesquisa utilizada em vetores e ou matrizes, em modo sequencial, elemento por elemento.
A Busca Binária: método mais eficiente do que o sequencial, mas para funcionar necessita que o vetor esteja ordenado.



A Busca Sequencial tem como problema o tempo que dura para achar uma informação, pois caso exista uma quantidade grande, ele vai percorrer todas as informações até achar o que está sendo procurado, já a Busca Binária depende de estar organizada, pois caso as informações estejam desencontradas, os dados não poderão ser localizados com a Busca Binária.

Exemplo de Busca Sequencial utilizando o VisualG:

```
algoritmo "busca"
var
opc: inteiro
n: inteiro
dadosvet: vetor[1..10] de inteiro
inicio
dadosvet[1] < -5
dadosvet[2] < -2
dadosvet[3] < -7
dadosvet[4] < -10
dadosvet[5] < -1
dadosvet[6] < -3
dadosvet[7] < -6
dadosvet[8] < -4
dadosvet[9] < -9
dadosvet[10] < -8
opc < -0
n < -1
enquanto opc=0 faca
  se dadosvet[n]=7  entao
    escreva("A posição do valor encontrado é:")
    escreva(n)
    opc < -1
   fimse
  n < -n+1
  fimenquanto
fimalgoritmo
```

Neste algoritmo, utilizamos o Enquanto e o Se para fazer uma varredura e uma comparação para buscar a informação desejada. Neste caso, o sistema apresenta a localização da posição da informação.

17 - LISTAS

> A estrutura que permite representar um conjunto de dados de forma a preservar a relação de ordem linear (ou total) entre eles é a lista linear. Uma lista linear é composta de nós, os quais podem conter, cada um deles, um dado primitivo ou um dado composto. (VELOSO, P. SANTOS, C., AZEREDO, P., FURTADO, A., 1983,79)

17.1 - LISTA DE COMPRAS

- Macarrão
- Molho de tomate
- Pão
- Queijo
- Presunto

Permite inserir e remover dados em qualquer ponto da lista.

17.1.2 - LISTA LIGADA

A Lista Ligada é uma forma de lista que possui ligação entre os itens pertencentes nela. Como exemplo temos:

Item **-->** item 2 **-->** item 3 **-->** item 4 **-->** itens...

Esta lista facilita a busca e a inserção de novas informações, como no exemplo da Lista de Compras, você pode retirar itens e inseri-los a hora que desejar, o mesmo acontece com a lista normal.

18 - PILHAS

Pilha é uma forma de armazenar memória em blocos empilhados um a um. Esses blocos são empilhados na ordem a, b, c e desempilhados na ordem c, b, a. Ou seja, quem foi empilhado por último será o primeiro a ser desempilhado. A analogia com uma pilha de pratos é óbvia e simples de imaginar.

As pilhas são listas, nas quais é aplicada uma disciplina de acesso, sendo o último a entrar é o primeiro a sair oi LIFO (Last In, First out).

18.1 - INCLUSÃO

A inclusão de dados na pilha deve ser feita sempre por cima, como em uma pilha de pratos, desta forma, sempre o último a entrar está na parte superior.

18.2 - REMOÇÃO

A remoção dos dados na pilha é sempre feita do último para o primeiro, tomando sempre cuidado no acesso.

18.3 - USO

Histórico de comandos: Normalmente, tudo o que você faz fica gravado em um histórico. Estes comandos estão em uma pilha, tanto que para acessá-los, devemos ir do último ao primeiro.

19 - FILAS

As filas são estruturas de dados que se comportam como uma fila do mundo real, ou seja, utilizam a técnica do primeiro a entrar é o primeiro a sair ou FIFO (First In, First Out).

19.1 - INSERÇÃO E REMOÇÃO

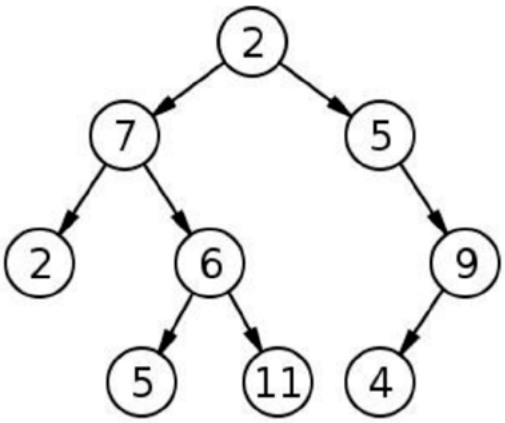
Na fila o elemento novo fica no final, ou seja, toda inserção é feita no ponto final, já a remoção é feita no início da fila. Como em uma fila de banco, o primeiro que chega deve ser o primeiro a ser atendido, ou seja, o primeiro a ser removido da fila.
Como utilizar?

Impressão **-->** Fila de impressão
Rede **-->** Fila de dados

20 - ÁRVORES

Estrutura de nós com uma base ou raiz.

A árvore é uma lista em que cada elemento possui dois ou mais nós sucessores, porém todos os nós possuem apenas um antecessor.



A estrutura de pastas do Windows e de diretórios do Linux é na verdade uma árvore.

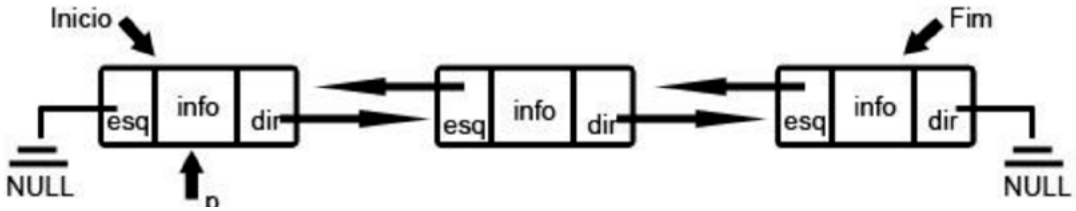
21 - OUTRAS ESTRUTURAS

Existem outras estruturas, algumas menos conhecidas que outras, mas com maiores complexidades.

21.1 - LISTA DUPLAMENTE ENCADEADA

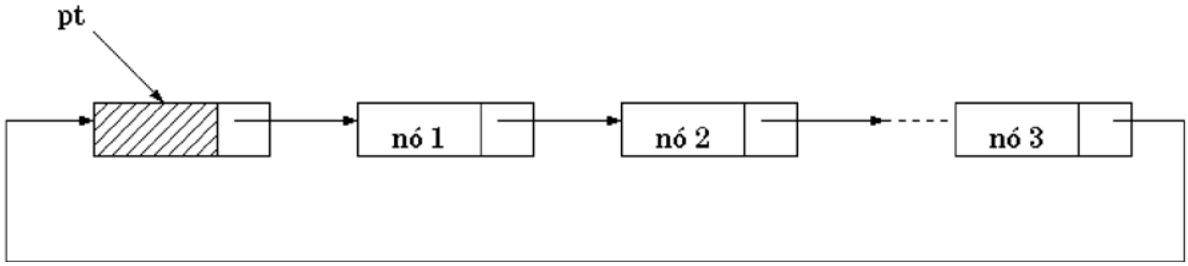
Na aula de lista, foi explicado que a lista tem a vantagem de poder inserir e remover os dados na ordem em que desejar.

A Lista Duplamente Encadeada é semelhante à Lista Ligada, com a diferença de ter ligação para frente e para traz, sendo mais fácil efetuar buscas.



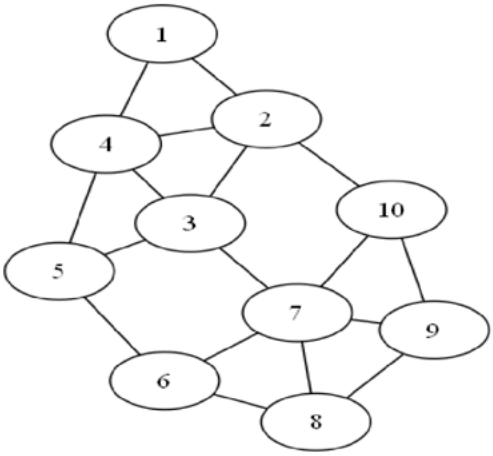
21.2 - LISTA CIRCULAR

A Lista Circular possui ligação do último iten com o primeiro.



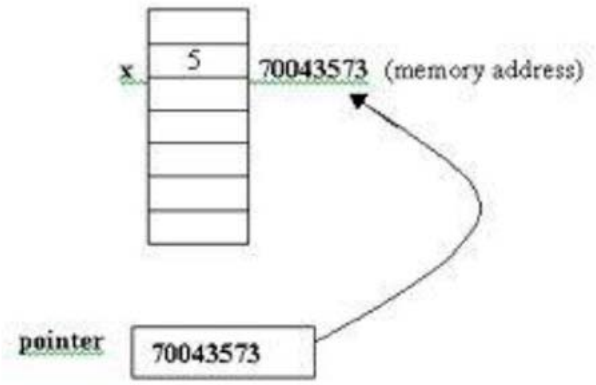
21.3 - GRAFOS

Os grafos são estruturas semelhantes às árvores, mas possuindo diversas ligações. Esta estrutura é largamente utilizada em sistemas comerciais e relações comerciais, redes de relacionamento como Orkut, Facebook, etc.



22 - PONTEIRO

O ponteiro nada mais é do que uma variável que guarda o endereço de outra variável.



Para trabalhar com Listas Ligadas, Duplamente Ligadas, Circular, etc., é necessária a utilização de ponteiros.

"Basicamente, um ponteiro é uma representação simbólica de um endereço".
Razões para uso do ponteiro:

1. Manipular matrizes mais facilmente e movimentar entre estruturas.
2. Velocidade.

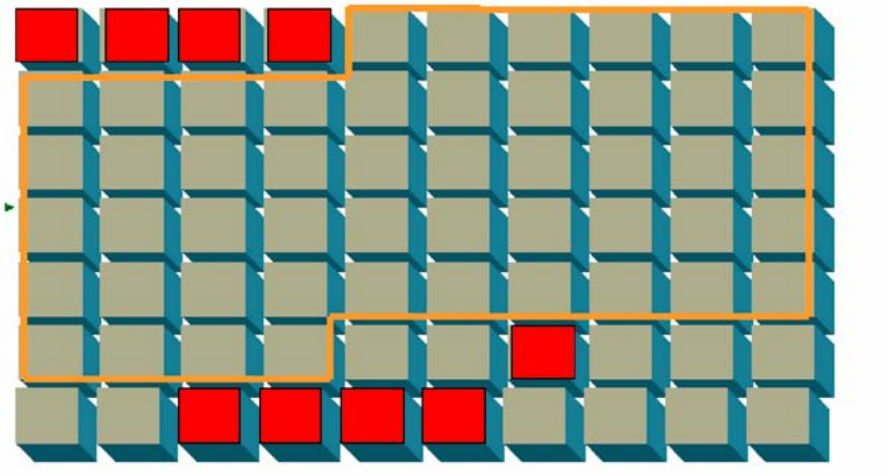
Exemplo:
A: inteiro
A < - 10
B: ponteiro
B < - A

O valor de B na verdade é o endereço de A.

23 - ALOCAÇÃO DINÂMICA

Alocação Dinâmica é o processo de solicitar e utilizar memória durante a execução de um programa.

O principal ponto é a redução de consumo de memória.



Quando utilizar?

Imagine o seu sistema de agendas, mas você não sabe quantas pessoas vai inserir na agenda, logo o mais adequado é utilizar a Alocação Dinâmica, evitando-se assim o excesso de uso de memória.

- Benefícios

- Criação de variáveis com o tamanho correto.
- Utilização correta da memória.
- Evita buffer de overflow.

24 - ORIENTAÇÃO A OBJETOS

É uma abordagem para a construção de software, que apresenta soluções para os problemas clássicos de desenvolvimento de sistemas da informação.

O conceito básico de Orientação a Objetos é que todo software deve ser construído a partir de componentes padronizados e reutilizáveis.

- Benefícios

- Produtividade
- Qualidade
- Manutenibilidade
- Aderência ao negócio

24.1 - O QUE É UM OBJETO?

Um objeto é qualquer coisa, real ou abstrata, a respeito da qual armazenamos dados e os métodos que os manipulam.

Saiba Mais!

www.fgrweb.com.br
www.portaldaprogramacao.com

Respostas dos exercícios

Capítulo 1

1 – Refine o algoritmo de Troca de Pneus.

Para iniciar vamos:

- Parar o carro.
- Puxar o freio de mão.
- Ligar o pisca-alerta.
- Abrir o porta-malas.
- Retirar o triângulo.
- Posicionar a 10 metros do carro.
- Retornar ao porta-malas.
- Retirar a chave de rodas.
- Retirar o macaco.
- Retirar o estepe.
- Levantar o carro com o macaco.
- Desapertar os parafusos da roda.
- Retirar o pneu furado.
- Colocar o estepe.
- Recolocar os parafusos.
- Apertar os parafusos.
- Baixar o carro com o macaco.
- Guardar o pneu furado no porta-malas.
- Guardar a chave de rodas no porta-malas.
- Guardar o macaco no porta-malas.
- Buscar o triângulo.
- Guardar no porta-malas.
- Entrar no carro.
- Desligar o pisca-alerta.
- Dar seta.
- Soltar o freio de mão.
- Finalizar.

2- Crie um algoritmo para calcular a média de um aluno, sendo que a média é igual à nota do primeiro bimestre mais a nota do segundo bimestre dividido por dois.

```
algoritmo “media”
var
valor1, valor2: inteiro
media: real
inicio
escreva (“Digite o primeiro valor”)
leia (valor1)
escreva (“Digite o segundo valor”)
leia (valor2)
media < - (valor1 + valor2)/2
se (media > 6) entao
escreval (“Aluno aprovado”)
senao
escreval (“Aluno reprovado”)
fimse
escreva (“Sua media é: “)
```

escreval (media)
fimalgoritmo

Capítulo 2

1 – No modelo clássico/cascata qual a função da fase de Implementação?

No momento da implementação é a hora de codificar seu programa, envolve ainda a integração dos módulos.

2 - Qual a principal diferença entre o Modelo Clássico e o Modelo Ágil?

A principal diferença é que no Modelo Ágil os testes são feitos durante cada fase do desenvolvimento, diferente do Modelo Clássico, em que a manutenção começa ao concluir a implantação.

Capítulo 3

1 – Crie um algoritmo que faça a multiplicação de 4 números.

algoritmo "multiplicação"
var
inicio
escreva (8*10*2*3)
fimalgoritmo

2 - Crie um algoritmo que exiba a média entre os números 8, 4, 6 e 9;

algoritmo "media"
var
inicio
escreva ((8+4+6+9)/4)
fimalgoritmo

Capítulo 4

1 – Crie um algoritmo que faça a multiplicação de 4 números, mas agora sendo solicitado os dados do usuário.

algoritmo "multiplicação"
var
a,b,c,d: inteiro
inicio
escreva ("Digite o primeiro valor")
leia (a)
escreva ("Digite o segundo valor")
leia (b)
escreva ("Digite o terceiro valor")
leia (c)
escreva ("Digite o quarto valor")
leia (d)
escreva (a*b*c*d)
fimalgoritmo

Capítulo 5

1 – Com base no que você já aprendeu, faça um algoritmo utilizando condicional para dizer se o aluno está aprovado ou reprovado.

```
algoritmo "multiplicação"
var
a,b: inteiro
media: real
inicio
escreva ("Digite a primeira nota")
leia (a)
escreva ("Digite a segunda valor")
leia (b)
media < - (a+b)/2
se (media<6) entao
escreval ("Aluno reprovado")
senao
escreval ("Aluno aprovado")
fimse
escreval ("A media do aluno foi: ")
escreva (media)
fimalgoritmo
```

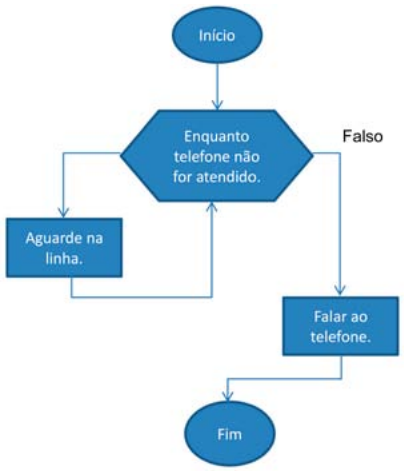
Capítulo 6

1 – Utilize o exemplo que criamos de Time de Futebol e complete fazendo uma comparação com os times do seu estado.

```
algoritmo "times"
var time: caractere
inicio
escreva ("Entre com o nome de um time de futebol: ")
leia (time)
escolha time
caso "Flamengo", "Fluminense", "Vasco", "Botafogo"
    escreval ("É um time carioca.")
caso "São Paulo", "Palmeiras", "Santos", "Corinthians"
    escreval ("É um time paulista.")
outrocaso
    escreval ("É de outro estado.")
fimescolha
fimalgoritmo
```

Capítulo 7

1 – Com base no fluxograma apresentado, crie um fluxo para fazer uma ligação aguardando ser atendido.



Capítulo 8

1 – Agora que você já conhece as estruturas, pratique criando um algoritmo que faça cadastro em uma agenda de 10 pessoas.

```
algoritmo “agenda”
var
nome: caractere
contador: inteiro
inicio
escreval (“Agenda”)
para contador de 1 ate 10 faca
escreval (“Digite o nome”)
leia (nome)
escreva (“Contato numero: “)
escreval (contador)
escreval (nome)
contador < - contador +1
fimpara
fimalgoritmo
```

Capítulo 9

1 – Agora que você já conhece o vetor, pratique criando um algoritmo que faça cadastro em uma agenda de 10 pessoas em vetor.

```
algoritmo “agenda”
var
nome: vetor[1..10] de caractere
contador: inteiro
inicio
escreval (“Agenda”)
para contador de 1 ate 10 faca
escreval (“Digite o nome”)
leia (nome[contador])
escreva (“Contato numero: “)
escreval (contador)
escreval (nome[contador])
contador < - contador +1
fimpara
fimalgoritmo
```

Capítulo 10

1 - Agora que você já conhece o vetor, pratique criando um algoritmo que faça cadastro em uma agenda de 10 pessoas em uma matriz com nome e e-mail.

```
algoritmo “agenda”
var
nome: vetor[1..10,1..2] de caractere
contador: inteiro
inicio
escreval (“Agenda”)
para contador de 1 ate 10 faca
escreval (“Digite o nome”)
leia (nome[contador,1])
escreval (“Digite o e-mail”)
```

```
leia (nome[contador,2])
escreva (“Contato número: “)
escreval (contador)
escreva (“O nome é:”)
escreval (nome[contador,1])

escreva (“O e-mail é:”)
escreval (nome[contador,2])
contador < - contador +1
fimpara
fimalgoritmo
```

Capítulo 11

1 – Faça um algoritmo com função para soma e subtração.

```
algoritmo “fparametro”
var
   n,m,res, opc: inteiro
inicio
funcao soma (x,y: inteiro): inteiro
inicio
retorne x + y
fimfuncao
funcao subtracao (x,y: inteiro): inteiro
inicio
retorne x - y
fimfuncao
escreva (“Digite 1 para soma ou 2 para subtração”)
leia (opc)
escreva (“Digite o primeiro valor”)
leia (n)
escreva (“Digite o seguindo valor”)
leia (m)
escolha (opc)
caso 1
    res < - soma (n,m)
caso 2
    res < - subtracao (n,m)
outrocaso
        escreva (“Não foi escolhido nem soma e  nem subtração”)
fimescolha

escreva(res)
fimalgoritmo
```

Capítulo 12

1) Complete o segundo algoritmo deste capítulo com multiplicação e divisão.

```
algoritmo “fparametro”
var
  n,m,res, opc: inteiro
  res1: real
inicio
funcao soma (x,y: inteiro): inteiro
inicio
retorne x + y
```

```
fimfuncao
funcao subtracao (x,y: inteiro): inteiro
inicio
retorne x - y
fimfuncao
funcao multiplicacao (x,y: inteiro): inteiro
inicio
retorne x * y
fimfuncao
funcao divisao (x,y: inteiro): real
inicio
retorne x / y
fimfuncao

escreva (“Digite 1 para soma, 2 para subtração, 3 para multiplicação ou 4 para divisão”)
leia (opc)
escreva (“Digite o primeiro valor”)
leia (n)
escreva (“Digite o seguindo valor”)
leia (m)
escolha (opc)
caso 1
    res < - soma(n,m)
caso 2
    res < - subtração (n,m)
caso 3
    res < - multiplicação (n,m)
caso 4
    res1 < - divisão (n,m)
    escreva(res1)
outrocaso
        escreva (“Não foi escolhido nem soma nem subtração”)
fimescolha

escreva (res)
fimalgoritmo
```

Referências Bibliográficas

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de. *Algoritmos: Lógica para desenvolvimendo de programas.* 9. Ed. São Paulo: Érica, 2000.

QUINE, O. *O Sentido da Nova Lógica*. São Paulo: Editora da USP, 1942.

alfamacursos.com.br